Henna Ahoranta

VERKKOSIVUN TIEDONHARAVOINNIN AUTOMATISOIMINEN

Case: EnergiaGuru

Opinnäytetyö
CENTRIA-AMMATTIKORKEAKOULU
Sähkötekniikan koulutusohjelma
Huhtikuu 2017

TIIVISTELMÄ OPINNÄYTETYÖSTÄ

Centria-ammattikorkeakoulu	**Aika** Huhtikuu 2017	**Tekijä/tekijät** Henna Ahoranta
Koulutusohjelma Sähkötekniikka		
Työn nimi VERKKOSIVUN TIEDONHARAVOINNIN AUTOMATISOIMINEN. Case: EnergiaGuru		
Työn ohjaaja Hannu Puomio		**Sivumäärä** 33 + 6
Työelämäohjaaja Virve Sorvoja		

Työssä tutkittiin datan hakemista verkkosivulta PHP-ohjelman avulla. Haetusta datasta etsittiin tarvittavat tiedot ja ne tallennettiin omaan tietokantaan. Työssä tutkittiin myös ohjelman ajastusta.

Teoriaosassa esiteltiin verkkosivun tietoharavointia, säännöllisiä lausekkeita, PHP-ohjelman muodostamisessa tarvittuja luokkakirjastoja sekä ajastusmahdollisuuksia. Lisäksi esiteltiin apuna käytettyä Wireshark-ohjelmaa sekä sen hyödyntämistä oikeanlaisen HTTP-pyynnön muodostamisessa ja ohjelman toiminnan tutkimisessa.

Käytännön osassa ohjelma toteutettiin PHP:llä. Aluksi HTTP-viestejä analysoitiin Wiresharkilla ja luotiin sen pohjalta HTTP-pyyntö PHP/CURL-kirjaston avulla. Palvelimelta vastaukseksi saatua tekstiä käsiteltiin sopivaan muotoon ja eroteltiin halutut arvot säännöllisten lausekkeiden avulla. Suodatettu data lähetettiin tietokantaan ja testattiin tehtävän ajastusta.

Lopuksi pohdittiin mahdollisia parannuskohtia sekä esitettiin parannus- ja jatkokehitysehdotuksia ohjelman käytettävyyden, luotettavuuden ja konfiguroinnin parantamiseksi.

Osa opinnäytetyön liitteistä on jätetty pois julkisesta versiosta.

Asiasanat
Automatisointi, cron, cURL, HTTP, MySQL, PHP, regex, web scraping, säännölliset lausekkeet, tehtävän ajastaminen, tietokanta, webharavointi.

ABSTRACT

Centria University of Applied Sciences	Date April 2017	Author Henna Ahoranta
Degree programme Electrical Engineering		
Name of thesis AUTOMATION OF WEB SCRAPING. Case: EnergiaGuru		
Instructor Hannu Puomio		**Pages** 33 + 6
Supervisor Virve Sorvoja		

The purpose of the thesis was to build a PHP script that could scrape data from a web page, extract the needed information from the data, insert the information into a database and automate the process to be executed once a day.

In the theoretical part web scraping, regular expressions, useful libraries and functions for constructing the PHP program and job scheduling options were studied. In addition, the packet analyzer program Wireshark, and how to use it for examining HTTP messages was introduced.

In the practical part a PHP script was written. At first, HTTP messages were analyzed with Wireshark and then an HTTP request was constructed with PHP/CURL library. The server's response was processed and filtered with regular expressions. Filtered data was inserted into the database and job scheduling was tested.

Finally, some ideas to improve and further develop the usability, reliability and configuration of the PHP script were discussed.

Key words
Automation, cron, cURL, database, HTTP, job scheduling, MySQL, PHP, regex, web scraping.

KÄSITTEIDEN MÄÄRITTELY

Cron	Unix-pohjaisilla käyttöjärjestelmillä käytettävä tehtävänajastuspalvelu
cURL	Ohjelmoinnissa käyttävä datan lähettämiseen tarkoitettu kirjasto, joka tukee useita Internet-protokollia
DNS	Domain Name System (DNS). Internet-protokolla, joka huolehtii verkkotunnuksien muuttamisesta IP-osoitteiksi
GNU GPL-lisenssi	GNU General Puplic License (GPL), vapaiden ohjelmistojen lisenssi. GNU GPL-lisenssillä julkaistuja ohjelmia saa kuka tahansa ladata, käyttää, kopioida, jakaa ja muokata vapaasti.
HTML	Hypertext Markup Language (HTML), Internet-sivujen kirjoittamiseen käytetty kuvauskieli
HTTP	Hypertext Transfer Protocol (HTTP), WWW-palvelinten ja -selainten käyttämä tiedonsiirtoprotokolla
MySQL	Tietokantaohjelmisto, joka on suosittu erityisesti web-ympäristössä
Pakettianalysaattori	Tietoliikenteen tutkimiseen käytettävä tietokoneohjelma
Palvelin	Engl. server. HTTP-pyyntöihin vastaava osapuoli
PHP	PHP: Hypertext Preprocessor, erityisesti web-ympäristössä käytetty ohjelmointikieli
Regex	Ks. säännölliset lausekkeet
Skripti	Komentosarja. Ohjelma, jota ei tarvitse kääntää ennen suorittamista
Säännölliset lausekkeet	Engl. regular expressions. Tekstinkäsittelytyökalu. Käytetään säännöllisten merkkijonojen määrittelemiseen
TCP	Transmission Control Protocol (TCP), OSI mallissa kuljetuskerroksessa toimiva tiedonsiirtoprotokolla (engl. Transport layer)
Unix	Unix on laitteistoriippumaton käyttöjärjestelmä
Verkkosivujen tiedonharavointi	Engl. web scraping. Tiedon etsimistä ja poimimista verkkosivuilta
Web-selain	Toimii HTTP-viestinnässä asiakasohjelmana. Esimerkiksi Internet Explorer, Google Chrome, Mozilla Firefox
WireShark	Ks. pakettianalysaattori

TAULUKOT

1 JOHDANTO

Opinnäytetyön tarkoituksena oli automatisoida tietokoneella suoritettava työtehtävä. Työtehtävä oli kerätä tietoa internetsivulta, eli kyseessä oli web-haravointi. Alun perin työtehtävä suoritettiin siten, että työntekijä meni nettiselaimen avulla tietylle nettisivulle, kopioi sieltä valintojensa perusteella muodostetun taulukon, poimi taulukosta tarvitsemansa tiedot ja tallensi ne tietokantaan. Opinnäyteyöni tarkoitus oli tutkia ja testata, miten tehtävä voitaisiin suorittaa PHP-skriptillä kokonaan ilman ihmiskäyttäjän apua. Ratkaisun oli tarkoitus kyetä toimimaan itsenäisesti, eli ohjelman oli tarkoitus käynnistyä automaattisesti ennalta määrättynä ajankohtana päivittäin.

Työprosessi alkoi web-haravointiprosessiin tutustumalla. Näin saatiin hahmoteltua työprosessin kulku, ja voitiin tutustua tarvittaviin työkaluihin ja -menetelmiin. Seuraavassa työvaiheessa perehdyttiin kohdesivustoon sekä HTTP-protokollaan. Aluksi täytyi selvittää, miten palvelimen kanssa kommunikointi ilman selainta toimii. Oli myös perehdyttävä dynaamisen sisällön muodostumiseen sekä HTTP-pakettien analysointiin. Tämän jälkeen siirryttiin tekstin muokkaamiseen ja suodattamiseen säännöllisten lausekkeiden avulla sekä tietokannan kanssa kommunikointiin. Lopuksi selvitettiin mahdollisia ratkaisuja tehtävän ajastusta varten.

Kappaleet pyrkivät etenemään samassa järjestyksessä kuin itse työn vaatima prosessi sekä siten, että liikutaan yleisemmistä käsitteistä yksityiskohtiin työn edetessä. Johdantoa ja yrityksen esittelyä seuraavat neljä kappaletta esittelevät työn kulkua sekä työn toteutuksessa käytettäviä työmenetelmiä sekä työkaluja ja tarvittavia käsitteitä. Nämä luvut muodostavat opinnäytetyön teoriapohjan. Seitsämännessä kappaleessa käydään läpi pala palalta käytännön työprosessi. Tässä kappaleessa kerrotaan yksityiskohtaisesti, miten ohjelma on koottu ja miten se toimii. Käytännön osuuden jälkeen pohditaan, millaisia jatkokehitysmahdollisuuksia toteutetulla ohjelmalla on. Viimeiseksi tehdään vielä yhteenveto työstä ja pohditaan kehitysmahdollisuuksia.

Työssä lähteinä käytetty aineisto on suurimmaksi osaksi englanninkielistä. Lähteinä on käytetty muun muassa ohjelmien, kuten Wiresharkin, cronin ja crontabin, sekä PHP-ohjelmointikielen käyttöohjeita ja nettidokumentaatioita. Lisäksi on hyödynnetty työn kannalta olennaisia asioita käsittelevää kirjallisuutta sekä Internet-lähteitä, kuten RFC-standardeja. The Internet Engineering Task Forcen (IETF) on kansain-

välinen organisaatio, jonka tehtävä on kehittää Internetin arkkitehtuuria ja toimintaa. Request for Comments (RFC) -julkaisut ovat IETF:n standardeja, jotka määrittelevät esimerkiksi Internet-protokollien toimintaa.

2 SUOMEN ENERGIANEUVONTA OY

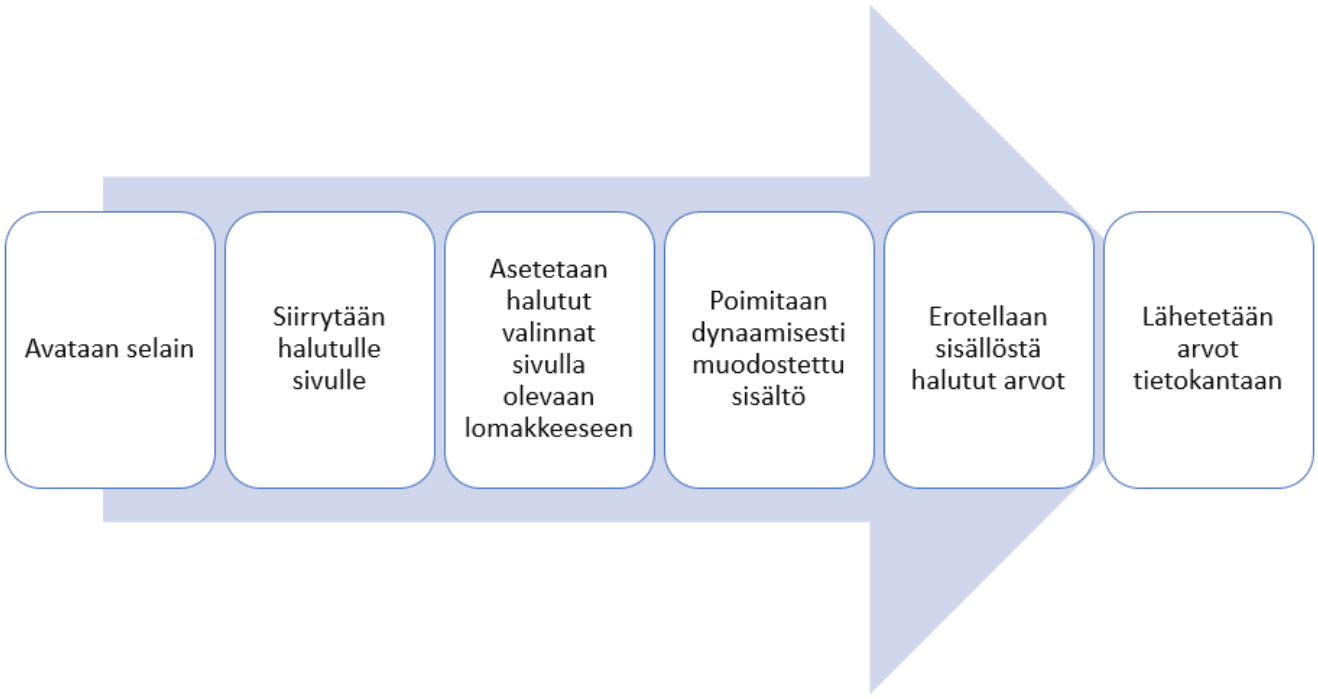
Suomen Energianeuvonta Oy on energianeuvontaan erikoistunut yritys, jonka kotipaikka on Alavieska. Suomen Energianeuvonta Oy tarjoaa EnergiaGuru-palvelua, joka auttaa asiakkaita vertailemaan sähkön hintaa ja kilpailukykyä sekä saavuttamaan säästöjä energiakuluissa. Palvelu perustuu muun muassa sähkömarkkinoiden hintojen seurantaan ja analysointiin. Energiaguru-palvelukonsepti on saanut tukea Keksintösäätiöltä ja kehityksessä on tehty yhteistyötä Vaasan ja Oulun yliopistojen sekä Centria Ammattikorkeakoulun kanssa. Energiaguru-palvelulle on haettu patenttia sekä tavaramerkkisuojaa. (EnergiaGuru a.)

Energiagurun asiakkaat ovat yrityksiä sekä yhteisöjä. Esimerkkeinä mainittakoon Bittium, UPM, Ojala Group, Pohjois-Pohjanmaan sairaanhoitopiiri sekä Ylivieskan kaupunki. Asiakkaiden energiankulutus on useimmiten luokkaa 6-30 GWh vuodessa, mutta myös yli 100 GWh luokan asiakkaita löytyy. (EnergiaGuru b.)

3 TIEDON HARAVOINTI JA PROSESSOINTI

3.1 Tiedonharavointi

Automatisoitavaa työtehtävää on kuvattu kuviossa 1.



KUVIO 1. Automatisoitava prosessi

Kyseessä on tiedonharavointiprosessi. Tiedonharavointi, englanniksi web scraping, tarkoittaa tiedon keräämistä internetsivuilta. Yleensä tietoa kerätään ja kootaan esimerkiksi tietokantaan, josta sitä voidaan käyttää tai analysoida myöhemmin. Tiedonharavointiin käytetään tietokoneohjelmia, mutta sitä voi tehdä myös käsin. Käsin haravointi onnistuu internetselaimen avulla. Käyttäjä siirtyy halutulle sivustolle, etsii ja kopioi haluamansa tiedon ja liittää sen haluamaansa tietokantaan tai tiedostoon. Robotit yleensä käyttävät funktioita tai kirjastoja, jotka mahdollistavat HTTP-viestien lähettämisen. Pääpiirteiltään robotin suorittama prosessi on sama kuin ihmisen: etsitään ja kopioidaan haluttu resurssi, erotellaan siitä halutut tiedot ja siirretään ne tietokantaan.

Tietoa haravoivien robottien odotetaan noudattavan sivustojen robots.txt-tiedostoissa määrättyjä sääntöjä. Sivustot voivat robots.txt-tiedoston avulla ilmoittaa, jos ne eivät salli jonkin resurssin käyttöä au-

tomaattisilta prosesseilta. On myös hyvä ilmoittaa HTTP-pyynnön otsikkotiedoissa User-Agent-kohdassa, että kyseessä on robotti. User-Agent-tiedon avulla serveri saa selville, millainen ohjelma, esimerkiksi mikä selain tai robotti, pyrkii käyttämään sivuston sisältöä. Robottien tulisi ajastaa tiedonharavointitehtävänsä siten, etteivät ne ylikuormita palvelimia. (Glez-Peña, Lourenco, Lopez-Fernandez, Reboiro-Jato & Fdez-Riverola 2013.)

3.2 Verkkosivun dynaaminen sisältö

Prosessia automatisoitaessa on lähdettävä liikkeelle sivustolle siirtymisestä sekä lomakkeen lähettämisestä. Lomakkeen lähettäminen muuttaa sivun sisältöä. Tästä voidaan päätellä, että kyseessä on dynaaminen sisältö. Verkkosivun dynaaminen sisältö on sisältöä, joka muodostetaan dynaamisesti jonkin skriptin avulla. Skriptin lopputulokseen voi vaikuttaa esimerkiksi lähetetty lomake, käyttäjän identiteetti tai aikaleima.

Dynaamisen sisällön muodostumista voidaan tutkia pakettianalysaattoriohjelmalla. Näin saadaan selville se prosessi, joka piilee selaimen ja palvelimen välillä lomaketta lähettäessä ja dynaamista sisältöä muodostettaessa.

3.3 Tekstin prosessointi ja säännölliset lausekkeet

Sisällön prosessointi voidaan aloittaa tarkastelemalla, millä logiikalla ihmiskäyttäjä poimii haluamansa arvot. Käyttäjä saattaa etsiä esimerkiksi tiettyjä asiasanoja, yksiköiden tunnuksia, tiettyä tiedostoformaatin päätettä tai tietyssä muodossa esiintyviä lukuja tai päivämääriä. Jos sisällöstä etsittäisiin tiettyjä arvopareja, kuten tuotteen nimi ja hinta tai lämpötila ja viikonpäivä, voisi olla tarpeen myös määritellä, miten tiedetään, mitkä arvot kuuluvat yhteen.

Säännöllinen lauseke, englanniksi regular expression tai lyhemmin regex, on tekstin käsittelyyn tarkoitettu työkalu. Säännöllinen lauseke on lauseke, joka määrittää säännöt halutulle merkkijonolle. Sitä voidaan käyttää esimerkiksi tietyt säännöt noudattavan merkkijonon löytämiseen, kopioimiseen, tarkistamiseen tai korvaamiseen jollain toisella merkkijonolla. Säännöllisiä lausekkeita tarvitaan muun mu-

assa silloin, kun halutaan etsiä esimerkiksi jotain tiettyä sanaa tai vaikkapa lämpötilatietoja pitkästä tekstistä tai suuresta määrästä dataa. Esimerkiksi yhteystietolomakkeessa annetun sähköpostiosoitteen muodon oikeellisuus voidaan tarkistaa säännöllisten lausekkeiden avulla.

Säännöllisiä lausekkeita muodostaessa on tärkeä tutkia sekä kohdetekstiä että haluttuja osumia ja pyrkiä tunnistamaan niistä tiettyjä säännönmukaisuuksia. Tärkeää on tunnistaa säännönmukaisuuksia, jotka erottavat halutut osumat muusta tekstistä. Hyödyllisiä säännönmukaisuuksia voi tilanteesta riippuen olla esimerkiksi numeroiden, kirjainten tai muiden merkkien määrä tai järjestys tai tietyn merkin tai merkkijonon esiintyminen. Säännönmukaisuuksia pyritään tämän jälkeen kuvaamaan säännöllisen lausekkeen mukaisella syntaksilla.

Säännölliset lausekkeet voivat myös sisältää niin kutsuttuja alilausekkeita. Alilausekkeet eroavat päälausekkeista siten, että otetaan huomioon lauseketta vastaavia osumia etsittäessä, mutta niitä ei välttämättä sisällytetä palautettavaan tulokseen. Alilausekkeet voivat esiintyä ennen tai jälkeen varsinaisia päälausekkeita, tai niiden sisällä. Esimerkiksi säännöllinen lauseke `(?<=foo)bar(?=foo)` sisältää sekä päälauseen `bar`, että ennen ja jälkeen päälausetta esiintyvät alilausekkeet `(?<=foo)` ja `(?=foo)`. Tämä säännöllinen lauseke etsisi tekstistä kohdan, jossa esiintyy tekstijono *foobarfoo*, mutta tulostaisi pelkästään sanan *bar*. Vaikka alilausekkeiden tuloksia ei tulosteta, on niiden silti ehdottomasti täytyttävä. Osumaa ei tulisi, jos tekstistä löytyisi vain sana *bar* tai *foobar* tai *barfoo*, vaan osuman saamiseksi kaikkien lausekkeiden ehtojen tulisi täyttyä täysin.

Säännöllisiä lausekkeita suunniteltaessa on hyvä ottaa huomioon määrittelyn tarkkuus. Säännöllisen lausekkeen tarkkuus tai epätarkkuus voi vaikuttaa esimerkiksi tarkistusnopeuteen sekä siihen, kuinka hyvin osumat vastaavat haluttua lopputulosta. Liian löyhästi määritellyn lauseen lopputulokseen voi epähuomiossa joutua vääriä osumia. Hyvin tarkasti määritellyn lauseen kohdalla lauseen ymmärtäminen ja vianetsintä voi käydä hankalaksi syntaksin monimutkaisuuden ja lauseen pituuden takia. Säännöllisten lauseiden muodostamisessa sekä niiden suoritusnopeuden ja tarkkuuden tutkimisessa voidaan käyttää apuna esimerkiksi osoitteesta https://regex101.com/ löytyvää Regular Expressions 101 -applikaatiota.

3.4 Tietokantaan lisääminen

Kun haluttu arvot on eroteltu, voidaan ne siirtää tietokantaan. Tässä työssä PHP-skripti suunniteltiin valmiin MySQL-tietokannan päälle. MySQL on erityisesti web-ympäristössä suosittu relaatiotietokanta,

joka on saatavilla vapaalla GNU GPL-lisenssillä (MySQL 5.7 Reference Manual 2017). Siirrettäessä tietoja tietokantaan oli tärkeää tietää, mihin tauluihin tietoja oltiin tallentamassa. Lisäksi oli tutkittava ja otettava huomioon taulujen väliset yhteydet.

Tietoja lisättäessä on hyvä ottaa huomioon kuinka usein tietoa tallennetaan, kuinka paljon ja kuinka kauan tietoa säilytetään sekä kuinka paljon tietoa kertyy tietokantaan. Esimerkiksi mikäli tietokannassa haluttaisiin säilyttää vain viimeisin tieto, olisi tarpeen automatisoida tietojen syöttämisen lisäksi myös vanhojen tietojen poistaminen.

3.5 Tehtävän ajastus

Viimeinen osa tiedonharavointiprosessin automatisointia on tehtävän ajastaminen. Tarkoituksena oli tutkia, miten tiedonharavointiprosessi voitaisiin suorittaa haluttuun aikaan ilman, että käyttäjän tarvitsisi sitä erikseen käynnistää halutulla hetkellä. Tehtävänajastuspalvelu on ohjelma, joka laukaisee tietyn tehtävän määrättynä ajankohtana automaattisesti.

4 WIRESHARK JA HTTP-PYYNTÖJEN TUTKIMINEN

Tässä työssä haluttiin automatisoida tehtävä, jonka ihminen oli aikaisemmin tehnyt käsin. Tehtävä tehtiin siten, että henkilö navigoi internetselaimen avulla tietylle nettisivulle, asetti siellä pudotusvalikosta haluamansa valinnat ja poimi sen jälkeen sivulta valintojensa perusteella muodostetun taulukon. Nyt haasteena oli suorittaa sama tehtävä ilman hiiren klikkauksia ja visuaalista selainta. Wireshark-ohjelmaa voitiin käyttää apuna, jotta saatiin selville, minkälaista selaimen ja palvelimen käymä tiedonvaihto todella oli.

Tutkittava prosessi oli:

1. Siirrytään haluttuun nettiosoitteeseen
2. Asetetaan halutut valinnat pudotusvalikoilla sekä valintaruudulla
3. Sivu päivittyy valintojen mukaan, saadaan haluttu taulukko näkyviin

Protokollatason viestien tarkasteluun voidaan käyttää pakettianalysaattoria. Pakettianalysaattorin avulla voidaan tarkastella, mitä viestejä selaimen ja palvelimen välillä liikkuu. Kun löydetään se selaimen lähettämä pyyntö, johon palvelin vastaa halutulla tavalla, voidaan muodostaa samanlainen HTTP-pyyntö PHP:n client URL -kirjaston avulla. Toisin sanoen tarkastellaan ja analysoidaan haluttua lopputulosta ja lähdetään sen jälkeen selvittämään, miten samanlainen lopputulos voitaisiin saada aikaan. Tätä lähestymistapaa kutsutaan takaisinmallinnukseksi.

4.1 HTTP-protokolla

HTTP on tietoliikenteen käsitemallissa applikaatiotasolla toimiva tiedonsiirtoprotokolla. Sen tarkoitus on siirtää applikaatiodataa. HTTP-protokollan mukaisessa kommunikaatiossa asiakasohjelma, englanniksi ”client”, eli esimerkiksi verkkoselain, lähettää palvelimelle HTTP-pyynnön. Selaimen lähettämästä HTTP-pyynnöstä käy ilmi esimerkiksi, mille url-osoitteelle pyyntö on tarkoitettu. Lisäksi pyyntö voi pitää sisällään esimerkiksi tiedon asiakasohjelmasta, eli ”User-Agentista”. Pyyntötyyppejä on olemassa useita erilaisia, joista GET ja POST ovat kyseessä olevan tiedonharavointiprosessin kannalta tärkeimmät. GET-pyynnöillä selain hakee muun muassa sivuston lähdekoodin sekä sivulla näkyvät kuvat.

POST-pyynnöt taas liittyvät usein esimerkiksi lomakkeisiin. Niille on ominaista, että asiakasohjelman on lähetettävä kohdesivustolle jotain tietoja, kuten esimerkiksi lomake vastauksineen. POST-pyynnön sisältö voi olla tyypiltään vaikkapa application/x-www-form-urlencoded, joka nimensä mukaisesti sisältää parametrina url-koodatun lomakkeen. (RFC7231 2014.)

Palvelin vastaa selaimen lähettämiin pyyntöihin asianmukaisilla, HTTP-protokollaa noudattavilla, vastauksilla. Yleisesti HTTP-vastaukset voidaan jakaa viiteen luokkaan statuskoodin ensimmäisen numeron perusteella.

1xx	Pyyntö vastaanotettu, jatketaan prosessia
2xx	Pyyntö vastaanotettu, ymmärretty ja hyväksytty onnistuneesti
3xx	Pyyntö uudelleenohjattu
4xx	Pyyntöä ei voitu täyttää asiakaspuolen virheen takia
5xx	Pyyntöä ei voitu täyttää palvelinvirheen takia

TAULUKKO 1. HTTP-vastauskoodit (RFC7231 2014.)

HTTP 200 tarkoittaa sitä, että haluttu sisältö löytyi ja se pystyttiin toimittamaan. Viesti sisältää pyyntöä vastaavan sisällön. HTTP 404 on yleinen virheilmoitustyyppi, ja se tarkoittaa, ettei pyynnön mukaista sisältöä löytynyt, eikä sitä näin ollen pystytty toimittamaan. HTTP-vastauksen sisältötyyppejä ovat esimerkiksi text/html ja text/css, jotka sisältävät koodia tai GIF ja PNG, jotka sisältävät kuvia. (RFC7231 2014.)

HTTP-protokolla toimii yksinkertaisimmillaan siten, että user agent, eli käyttäjä, lähettää palvelimelle jonkin pyynnön ja palvelin vastaa siihen. Kuvassa 1 on esitetty asiakasohjelman lähettämä yksinkertainen GET-tyyppinen pyyntö sekä palvelimen lähettämä 200 OK -vastaus. Asiakkaan pyynnön ensimmäiseltä riviltä käy ilmi, että pyynnön tyyppi on GET, haluttu tiedosto on hello.txt sekä käytetty protokolla on HTTP/1.1. Seuraavalla rivillä annetaan tieto user agentista, eli asiakkaan käyttämästä ohjelmasta. Tässä esimerkissä asiakas on käyttänyt pyynnön tekemiseen curl- ja libcurl-kirjastoja. User Agent voisi myös olla esimerkiksi hakurobotti, kuten Googlen Googlebot, tai tavallisen ihmiskäyttäjän käyttämä selain, kuten Google Chrome tai Mozilla Firefox. Pyynnön kolmannella rivillä määritellään, missä osoitteessa olevalla palvelimella haluttu tiedosto sijaitsee. Vastausviestissä määritellään muun muassa protokolla ja vastauksen tyyppi, päivämäärä, palvelin, sisällön pituus sekä sisällön tyyppi. Viimeinen rivi on asiakkaan pyyntöä vastaava sisältö, eli hello.txt tiedoston sisältämä teksti.

```
Client request:

  GET /hello.txt HTTP/1.1
  User-Agent: curl/7.16.3 libcurl/7.16.3 OpenSSL/0.9.7l zlib/1.2.3
  Host: www.example.com
  Accept-Language: en, mi


Server response:

  HTTP/1.1 200 OK
  Date: Mon, 27 Jul 2009 12:28:53 GMT
  Server: Apache
  Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
  ETag: "34aa387-d-1568eb00"
  Accept-Ranges: bytes
  Content-Length: 51
  Vary: Accept-Encoding
  Content-Type: text/plain

  Hello World! My payload includes a trailing CRLF.
```

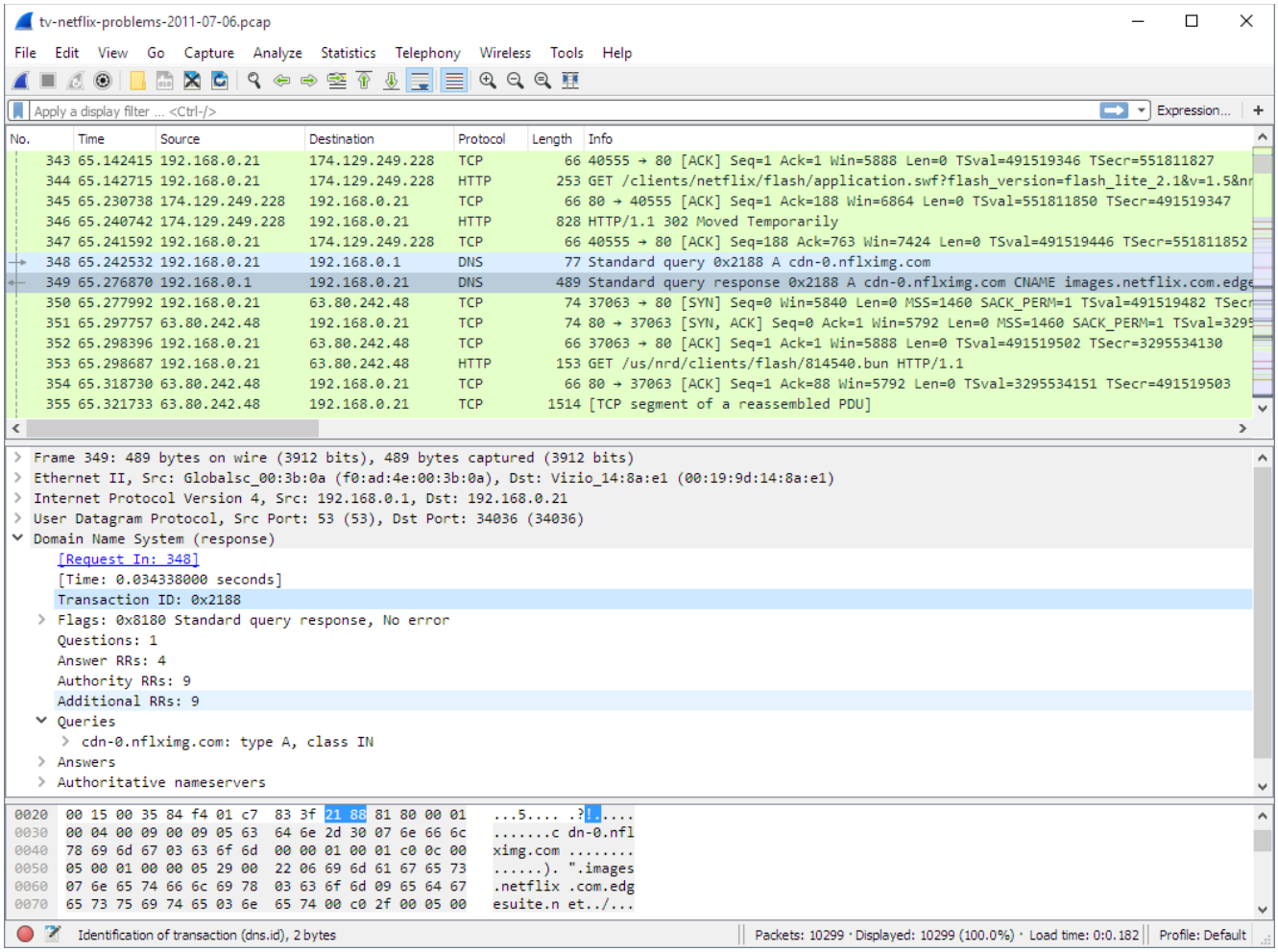
KUVA 1. Yksinkertainen esimerkki HTTP-pyynnöstä ja -vastauksesta (RFC7230 2014.)

4.2 Wireshark-pakettianalysaattori

Wireshark on tietokoneohjelma, jota käytetään verkkoviestinnän tutkimiseen. Wiresharkin avulla päästään näkemään verkon eri rajapintojen kautta kulkevaa tietoliikennettä, kuten TCP-, DNS- tai HTTP-protokollien mukaisia viestejä. Tällaista ohjelmaa kutsutaan pakettianalysaattoriksi. Wireshark on avoimella GNU GPL-lisenssillä lisensoitu, eli niin sanottu avoimen lähdekoodin ohjelmisto. Sen saa ladattua ja sitä pääsee käyttämään kuka tahansa täysin ilmaiseksi. Se toimii muun muassa Windows-, Linux-, OS X- ja UNIX-käyttöjärjestelmillä. Sitä voidaan hyödyntää esimerkiksi tietoliikenteen analysoinnissa, opetuksessa, vianetsinnässä sekä työkaluna kehitystyössä. (Wireshark User's Guide 2014.)

Kuvassa 2 näkyy Wireshark-ohjelman perusnäkymä. Näkymästä löytyy ylhäältä alaspäin valikko, työkalupalkki, rajauspalkki, pakettilistapaneeli, paketin tiedot-paneeli sekä statuspalkki. Työkalupalkista löytyvät muun muassa hainevän muotoinen aloituspainike sekä stop-painike, joiden avulla menossa oleva nauhoitus voidaan lopettaa ja aloittaa uusi nauhoitus. Rajauspalkissa olevaan tekstikenttään voidaan eritellä halutut protokollat. Nauhoitusnäkymä on lista kaikista nauhoitetuista paketeista. Listalta käy ilmi paketin numero, aika, lähettäjän osoite, vastaanottajan osoite, paketin protokolla, paketin pituus sekä muuta informaatiota. Muu informaatio voisi esimerkiksi HTTP-protokollan tapauksessa sisältää

tiedon pyynnön tyypistä ja sisällön tyypistä. Paketin tietoja voidaan tarkastella pakettilistan alapuolella. Paketti, jonka tietoja halutaan tarkastella, valitaan klikkaamalla haluttua pakettia pakettilistalta. Tietopaneelin avulla voidaan tarkastella kaikkia paketin sisältämiä tietoja.



KUVA 2. Wiresharkin perusnäkymä (Wireshark User's Guide 2014.)

Nauhoitettuja paketteja voidaan rajata protokollan mukaan. Rajaus voidaan tehdä joko ennen nauhoituksen aloittamista, nauhoituksen aikana tai nauhoituksen jälkeen. Protokollan rajaaminen olisi järkevää esimerkiksi silloin, kun ollaan kiinnostuneita tarkastelemaan vain tietyn protokollan tiedonvaihtoa. Wireshark nauhoittaa kaikkea halutun rajapinnan kautta kulkevaa liikennettä. Yleensä normaalia internetselailua nauhoittaessa paketteja tulee todella suuria määriä lyhyessä ajassa lukuisilta protokollilta. Tällöin esimerkiksi haluttuja HTTP-viestejä voi olla työläs etsiä kaikkien pakettien joukosta.

5 TYÖSSÄ KÄYTETYT OHJELMOINTIKIELET JA KIRJASTOT

5.1 PHP

PHP: Hypertext Preprocessor (PHP) on komentosarjakieli eli skriptikieli. Toisin kuin ohjelmointikieliä, komentosarjakieliä ei tarvitse kääntää ennen suorittamista, vaan niitä tulkitaan sitä mukaan, kun ohjelmaa suoritetaan. PHP:tä käytetään erityisesti web-ympäristössä dynaamisen sisällön luonnissa ja palvelinpuolen komentosarjakielenä, mutta sitä voidaan käyttää myös web-ympäristön ulkopuolella.

PHP/CURL

cURL, eli client URL, on kirjasto, jonka avulla PHP-kielessä voidaan muodostaa ja suorittaa muun muassa HTTP-protokollan mukaisia pyyntöjä. CURL ei kuulu PHP:n ytimeen, vaan se pitää erikseen kääntää, jotta sen voi ottaa käyttöön.

resource curl_init ([string $url = NULL])	Aloittaa uuden cURL-istunnon.
bool curl_setopt (resource $ch , int $option , mixed $value)	Asetetaan halutut tiedot cURL-yhteydelle. Hyödyllisiä valintoja ovat muun muassa:
- CURLOPT_POST	Asettaa HTTP-pyynnön tyypiksi POST.
- CURLOPT_POSTFIELDS	POST-pyynnön sisältämä data.
- CURLOPT_RETURNTRANSFER	Palvelimelta saatu vastaus tallennetaan muuttujaan. Ilman tätä valintaa palvelimelta saatu vastaus tulostetaan automaattisesti.
- CURLOPT_URL	URL-osoite, jolle pyyntö osoitetaan.
- CURLOPT_USERAGENT	Määrittelee tiedon HTTP-pyynnön User-Agent-kohtaan.
mixed curl_exec (resource $ch)	Suorittaa määritellyn cURL-istunnon.
void curl_close (resource $ch)	Lopettaa cURL istunnon ja vapauttaa kaikki resurssit.

TAULUKKO 2. Hyödyllisiä PHP/CURL-funktioita (PHP Manual 2017a.)

urlencode() ja urldecode()

Urlencode()- ja *urldecode()*-funktioita käytetään merkkijonojen url-koodaamiseen ja dekoodaamiseen. URL-koodaamista käytetään, kun merkkijonoja halutaan sisällyttää URL-osoitteeseen tai halutaan lähettää parametreja sivulta toiselle. Url-koodaaminen voi olla tarpeen esimerkiksi HTTP POST-pyyntöä lähettäessä.

Perl Compatible Regular Expressions (PCRE)

Perl Compatible Regular Expressions, lyhemmin PCRE, on PHP-kirjasto, joka on mahdollistaa säännöllisten lausekkeiden käytön PHP:llä. PCRE:tä käytetään muun muassa tekstin käsittelyyn ja tarkistukseen. PCRE:n säännöllisten lausekkeiden syntaksi muistuttaa Perlin säännöllisten lausekkeiden syntaksia. Funktion *preg_match()* avulla voidaan tarkistaa, löytyykö säännöllistä lauseketta vastaavia osumia. Tarvittaessa osumat voidaan tallentaa array-muuttujaan myöhempää käyttöä varten. *Preg_split()*-funktio jakaa tekstin array-muuttujaan. Jako suoritetaan säännöllistä lauseketta vastaavien osumien kohdalta. *Preg_replace()*-funktio korvaa säännöllistä lauseketta vastaavat osumat halutulla merkillä tai merkkijonolla. (Schrenk 2012, 50-52.)

Lisäksi on olemassa *preg_grep()*-funktio, joka ottaa parametreikseen säännöllisen lausekkeen sekä kohdetekstin array-muodossa. *Preg_grep()* palauttaa kokonaisuudessaan kaikki sellaiset array-elementit, joista löytyi säännöllistä lauseketta vastaava osuma. (PHP Manual 2017c.) Liitteestä 1 käy ilmi PCRE:n syntaksi.

Date/Time

Date/Time-funktioita käytetään muun muassa päivämäärän ja ajan muotoiluun sekä aikaleiman hakemiseen palvelimelta.

mysqli

MySQL Improved Extension, eli mysqli, on PHP:ssä tietokantayhteyksiin käytettävä työkalu. mysqli ei kuulu PHP:n ytimeen, vaan se tarvitsee erikseen kääntää, jotta sen voi ottaa käyttöön.

mysqli::$connect_error	Palauttaa viimeisimmän yhteysvirhesanoman.
mysqli::$error	Palauttaa viimeisimmän virhesanoman.
mysqli::query	Suorittaa annetun kyselylausekkeen.
mysqli_result::$num_rows	Tarkistaa, kuinka monta riviä kysely palautti.
mysqli_result::fetch_row	Hakee kyselyn tulokset array-elementteihin jaettuna.
mysqli::$insert_id	Hakee viimeisimmällä INSERT-kyselyllä viedylle tietueelle generoidun id:n.
mysqli::close	Sulkee tietokantayhteyden.

TAULUKKO 3. Hyödyllisiä mysqli-käskyjä (PHP Manual. 2017b.)

Uusi mysqli-yhteys voidaan määritellä seuraavalla tavalla:

```
$mysqli = new mysqli('localhost', 'my_user', 'my_pass-
word', 'my_db');
```

5.2 SQL

Structured Query Language (SQL) on kyselykieli, jonka avulla relaatiotietokannoista voidaan hakea, poistaa tai muuttaa tietoja. SQL-kielessä tietokantaan lisätään tietoa INSERT-käskyllä. Tietoja voi hakea SELECT-käskyllä.

- *SELECT* (halutut kentät) *FROM* (taulun nimi)
- *INSERT INTO* taulun nimi(halutut kentät) *VALUES* (kenttiin syötettävät tiedot)
- *WHERE* täytettävä ehto

WHERE-käskylle annetaan jokin looginen ehto, jonka avulla tuloksia halutaan rajata. Esimerkiksi käsky ”*WHERE* id > 5” hakisi taulusta kaikki tiedot, joiden id-kentän arvo olisi enemmän kuin 5. (MySQL 5.7 Reference Manual 2017.)

6 TEHTÄVÄN AJASTUS

Tehtävän ajastamisella tarkoitetaan sitä, että tietty toiminto asetetaan alkamaan itsenäisesti tiettynä ajankohtana. Internetissä ajastamiseen käytetään yleisesti muun muassa Cronia, joka on Unix-pohjainen ajastuspalvelu. Ajastuspalvelun käyttö voi olla tarpeellista esimerkiksi silloin, kun jokin yksinkertainen tehtävä vaatii säännöllistä suorittamista, eikä tehtävän suorittaminen varsinaisesti vaadi ihmisen läsnäoloa. Tällainen tilanne voisi olla vaikkapa tiedostojen varmuuskopiointi tai jonkin ohjelman suorittaminen. Kummatkin näistä voidaan hoitaa yksinkertaisella komentosarjakäskyllä.

Yksi vaihtoehto PHP-skriptin ajastamiselle on asentaa Cron joko verkkopalvelimelle tai jollekin UNIX-pohjaista käyttöjärjestelmää käyttävälle koneelle. Kaikki verkkopalveluntarjoajat eivät kuitenkaan salli cronin asentamista palvelimelle. Pöytäkoneelle asennettu Cron toimii vain silloin, kun kone on päällä, eli koneen pitäisi olla aina Cron-tehtävän suorittamisen aikaan käynnissä. (Ubuntu Documentation, Community Help Wiki 2016.)

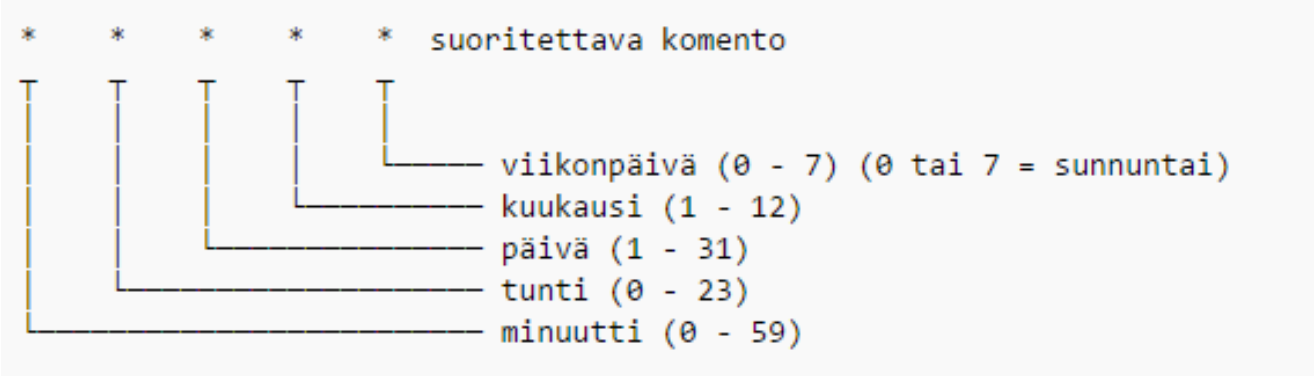
Toinen vaihtoehto on käyttää verkossa toimivia tehtävänajastuspalveluita (Ubuntu Documentation, Community Help Wiki 2016), kuten esimerkiksi Cronlessia ja Easycronia. Nämä tehtävänajastuspalvelut toimivat yleensä siten, että niille annetaan cron-skriptin osoite sekä määritellään aika, milloin tehtävät täytyy ajaa. Osa palveluista on ilmaisia, osa maksullisia. Muun muassa Cronlessilla sekä Easycronilla on olemassa sekä maksuton paketti että maksullisia paketteja. Paketit eroavat toisistaan muun muassa cron-tehtävien määrällä, tehtävien maksimisuoritustiheydellä sekä sillä, onko sähköposti-ilmoituksia tai virheilmoituslistoja saatavilla. Joissain tapauksissa maksuttomien pakettien tilaus täytyy säännöllisesti uusia, tai tehtävien ajastettu ajaminen katkeaa.

6.1 Cron ja crontab

Tehtävänajastus voidaan helposti asettaa UNIX-pohjaiselle käyttöjärjestelmälle. Ajastuspalvelu cron sekä sen hallinnoimiseen tarkoitettu crontab usein kuuluvat käyttöjärjestelmän oletustyökaluihin. Uusi cron-tehtävä voidaan asettaa crontabin avulla terminaalin kautta käskyllä `crontab -e`. Käsky avaa tekstitiedoston, jossa cron-tehtävät ja niiden suoritusajankohdat on määritelty. Tehtävät merkitään jo-

kainen omalle rivilleen. Ensimmäiseksi määritellään suoritusajankohta. Aikamäärittelyn jälkeen kirjoitetaan haluttu käsky samassa muodossa, kuin se annettaisiin suoraan terminaalissa. Esimerkiksi samalla koneella sijaitsevan php-skriptin voi suorittaa käskyllä `php -f /polku/haluttuun/tiedostoon.php`. Listan kaikista asetetuista cron-tehtävistä saa näkyviin käskyllä `crontab -l`. (Ubuntu Documentation, Community Help Wiki 2016.)

Suoritusajankohta määritellään kuvan 3 osoittamalla tavalla. Asteriski, eli *-merkki, tarkoittaa, että tehtävä suoritetaan jokaisella kerralla. Viikonpäivä-kohtaan pystytään määrittelemään haluttu viikonpäivä numeroilla 0-7 siten, että 0 vastaa sunnuntaita, 1 maanantaita, 2 tiistaita ja niin edelleen. Myös numero 7 vastaa sunnuntaita. Lisäksi voidaan määritellä haluttu kuukausi, päivä, tunti ja minuutti. Suoritusajankohtia voidaan määritellä useita. Suoritusalue voidaan määritellä joko niin, että kaikki halutut arvot listataan pilkulla eroteltuina tai antamalla lukualue. Lukualue merkitään erottamalla kaksi lukua toisistaan väliviivalla. Esimerkiksi `* 1-2 * * *` tarkoittaisi joka minuutti kello yhden ja kahden välillä jokaisen kuukauden jokaisena päivänä. Merkitsemällä luvun eteen vinoviiva saadaan määriteltyä suoritusintervalli. Esimerkiksi, jos jokin tehtävä haluttaisi ajaa jokaisena perjantaina kahdenkymmenen minuutin välein, merkittäisiin se `/20 * * * 5`.



KUVA 3. Ajan esitysmuoto crontabissa (Linux.fi 2016.)

Tehtävänajastusta voidaan haluta suojata tunnistamalla tehtävän laukaisija. Tehtävän laukaisijan tunnistaminen voi olla tarpeen esimerkiksi silloin, kun tehtävän suorittaminen väärään aikaan tai väärän tahon toimesta olisi haitaksi tehtävän suorittamiselle. Laukaisijan tunnistamiseen voitaisiin käyttää esimerkiksi tunnistusavainta tai HTTP-protokollan mukaista salausta käyttäjänimellä ja salasanalla.

6.2 EasyCron ja Cronless

Easycron tarjoaa yksityishenkilöille ilmaista pakettivaihtoehtoa. EasyCronilla maksuttomaan pakettiin kuuluu 200 Cron-tehtävän suoritusta päivässä. Tehtävien määrää ei ole rajoitettu, eli vaikka kaikki 200 ajoa voivat olla eri tehtäviä, tai vaihtoehtoisesti sama tehtävä voidaan ajaa useita kertoja. Esimerkiksi, jos kaksi tehtävää ajetaan 20 minuutin välein, tarkoittaa se 144 tehtävää, ja jos ajetaan kahdeksan eri tehtävää joka tunti, tarkoittaa se 192 tehtävää. Minimiväli yhden Cron-tehtävän ajolle ilmaisessa paketissa on 20 minuuttia, ja tehtävä aikakatkaistaan, mikäli se kestää yli 5 sekuntia. Mikäli tehtävän ajo epäonnistuu kahdesti, Easycron lopettaa sen ajamisen. Käyttäjän on mahdollista nähdä lokissa kymmenen suoritettua ja kymmenen seuraavaksi suoritettavaa tehtävää. Sähköposti-ilmoitukset eivät kuulu ilmaiseen pakettiin, ja ilmainen paketti on uusittava kuukauden välein. (EasyCron.com.)

Cronlessin ilmaisella paketilla Cron-tehtäviä voi asettaa enintään viisi, ja jokaisen voi ajaa enintään kaksi kertaa päivässä. Cron-tehtävän ajo lopetetaan, jos se epäonnistuu sata kertaa. Käyttäjän on mahdollista nähdä ja ladata itselleen virheilmoitusloki, ja palvelun kautta on mahdollista lähettää sähköpostiviesti, jossa ilmoitetaan ajon onnistumisesta tai epäonnistumisesta. (Cronless.com.)

7 TYÖN TOTEUTUS

Tässä kappaleessa on kuvattu työn toteutusta. Prosessi on esitelty pala palalta.

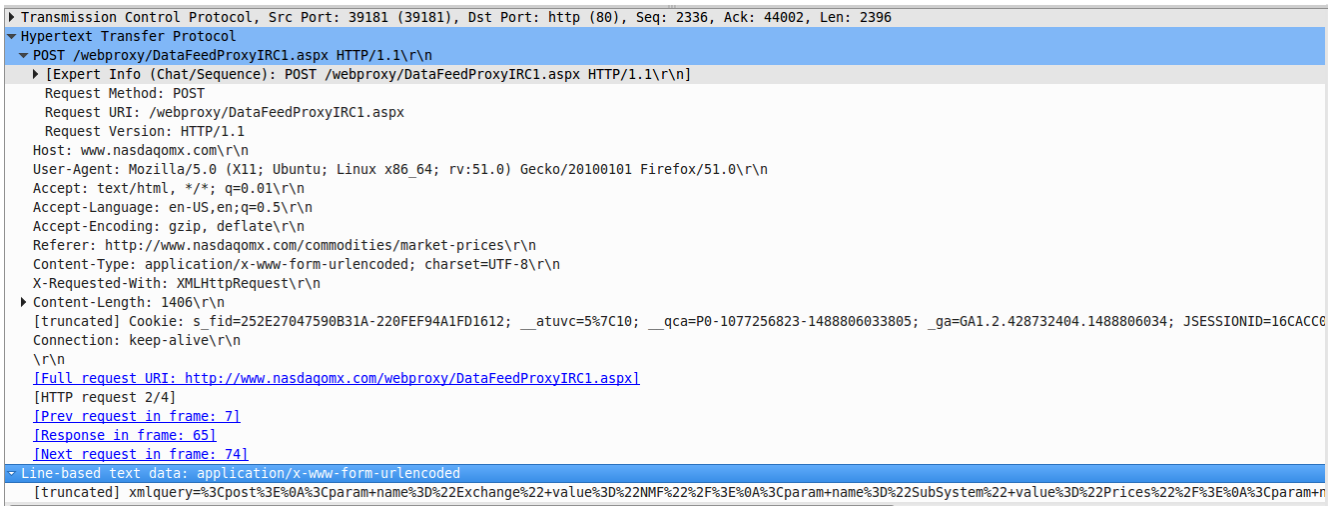
7.1 Tietoliikenteen analysointi WireSharkin avulla

Tehtävänannon mukaan haluttiin hakea sähkön pörssihintoja NASDAQin verkkosivuilta. Halutut hintatiedot sijaitsivat osoitteesta www.nasdaqomx.com/commodities/market-prices. Sivustoa tarkastelemalla voitiin huomata, että halutut hintatiedot löytyivät sivulta html-taulukosta. Taulukon sisältävät arvot vaihtuivat sivulla olevan lomakkeen mukaan. Lomakkeen avulla valittiin haluttu osakemarkkina, tuotetyypit sekä tuotteet. Lisäksi voitiin määritellä, näytetäänkö taulukossa vain toteutuneet kaupat, vai näytetäänkö kaikille tuotteille hinta siitä huolimatta, oliko niistä tehty kauppoja vai ei. Tämä määriteltiin valintaruudun avulla. Koska sisältö muuttui lomakkeen tietojen mukaan, voitiin päätellä kyseessä olevan dynaamista sisältöä. Täten taulukkoa ei ole koodattu suoraan lähdekoodiin, vaan se muodostetaan dynaamisen koodin avulla selaimen lähettämän HTTP-pyynnön mukaan.

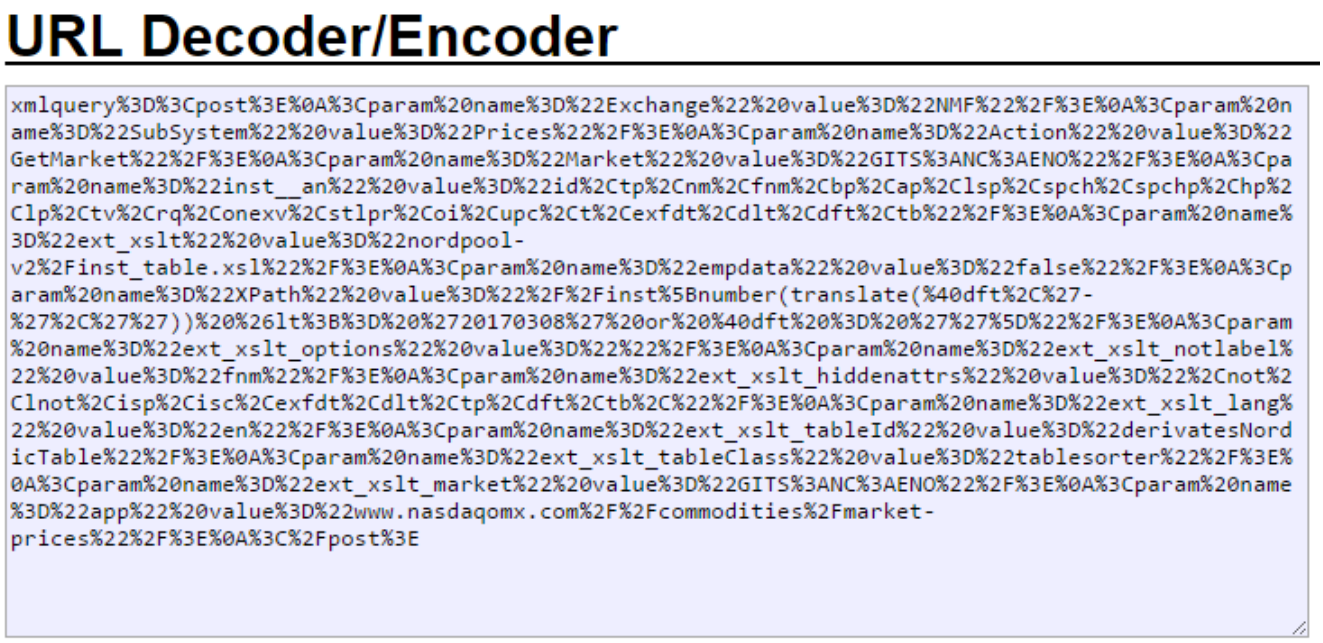
WireShark-nauhoituksen aloitusajankohdan valitseminen oli hyvin tärkeää analysointiprosessin kannalta. Mikäli nauhoitus olisi aloitettu jo ennen sivulle siirtymistä, olisi siinä ollut todella suuri määrä HTTP-paketteja, ja halutun viestinvaihdon löytäminen olisi ollut hidasta. Tällaisessa nauhoituksessa olisi näkynyt esimerkiksi koko sivun lähdekoodi sekä jokaisen yksittäisen kuvan pyyntö- ja vastauspaketit. Tällaiset paketit olisivat olleet tutkittavan asian kannalta täysin merkityksettömiä. Nauhoittaminen kannatti sen sijaan aloittaa vasta sivun lataamisen jälkeen ja juuri ennen hintatietotaulukon lomakkeen lähetystä. Näin saatiin aikaiseksi nauhoitus, jossa oli vain selaimen palvelimelle lähettämä POST-pyyntö ja vastausviestissä pelkkä taulukon koodi html-muodossa.

Kuten kuvan 4 nauhoituksesta käy ilmi, selain lähettää palvelimelle HTTP/POST-pyynnön muodossa application/x-www-form-urlencoded. Pyynnöllä haetaan resurssia /webproxy/DataFeedProxyIRC1.aspx. ”Host”-kohta kertoo, että resurssi sijaitsee palvelimella, jonka osoite on http://www.nasdaqomx.com/ . Näiden lisäksi olennaista on ”Line based text data: application/x-www-form-urlencoded” -otsikon alta löytyvä data. Tämä data on itseasiassa URL-koodattu HTML-lomake, johon on syötetty halutut arvot, kuten voidaan havaita kuvasta 6. Kuvista 5 ja 6 voidaan nähdä POST-pyynnön parametri

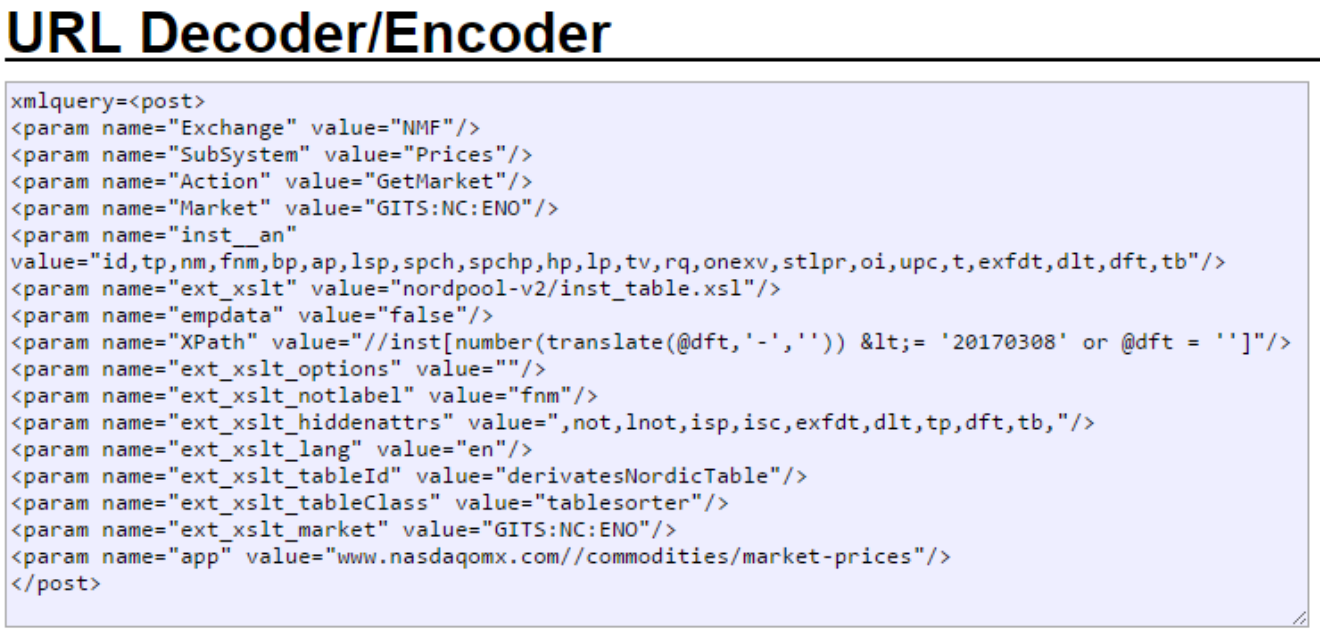
sekä koodattuna että dekoodattuna osoitteesta http://meyerweb.com/eric/tools/dencoder/ löytyvä URL-koodaimen avulla.



KUVA 4. Selaimen lähettämä HTTP-pyyntö



KUVA 5. POST-pyynnön parametri URL-koodattuna



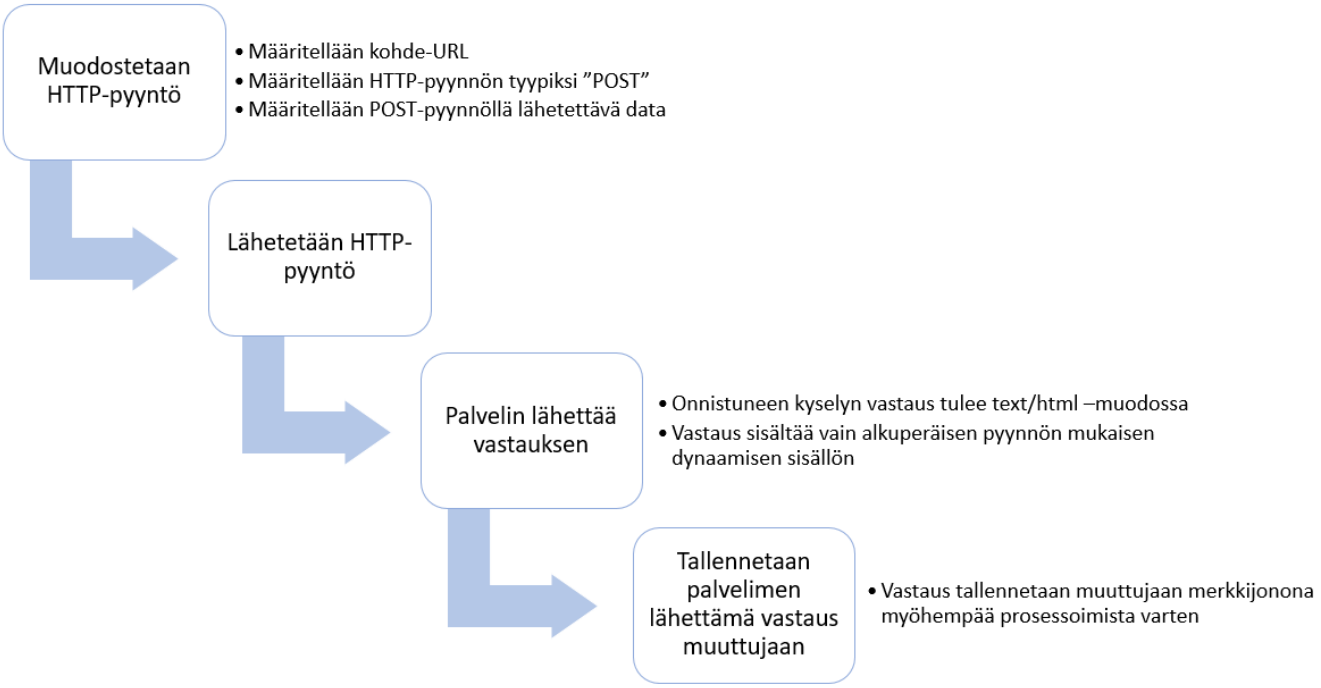
KUVA 6. POST-pyynnön parametri URL- dekoodattuna

7.2 HTTP-pyynnön lähettäminen PHP:llä

POST-pyynnön lähettämiseksi PHP:llä on pystyttävä määrittelemään vähintään kohdesivun URL, HTTP-pyynnön tyyppi, eli POST, sekä se parametri, joka POST-pyynnön välityksellä halutaan lähettää. Kaikki nämä tiedot ovat helposti löydettävissä Wireshark-nauhoituksesta. Nauhoituksen perusteella tiedettään, että halutut parametrit ovat:

- Kohdesivun URL: /webproxy/DataFeedProxyIRC1.aspx
- Pyynnön tyyppi: POST
- POST-pyynnön sisältö: xmlquery=%3Cpost%... dekoodattuna, kuten kuvassa 6

POST-pyynnön sisältö lisättiin koodiin url-dekoodatussa muodossa, ja se url-koodattiin *urlencode()*-funktiolla lähetystä varten. Koodiin voitaisiin lisätä myös User-Agent-tieto. User-Agentin ilmoittaminen ei ole lomakkeen lähetyksen ja halutun vastauksen saamisen kannalta oleellista, mutta se on kohteliasta. User-Agentin nimeksi määriteltiin tässä ”Testbot”. User-Agent -kenttässä voitaisiin haluttaessa myös ilmoittaa esimerkiksi url-osoite, josta löytyy tietoa botin omistajasta, toiminnasta ja tarkoituksesta. Nasdaqomx.com-sivuston robots.txt-tiedosto sijaitsee osoitteessa http://www.nasdaqomx.com/robots.txt. Tiedostosta käy ilmi, että hakemamme resurssi ei ole roboteilta kielletty.



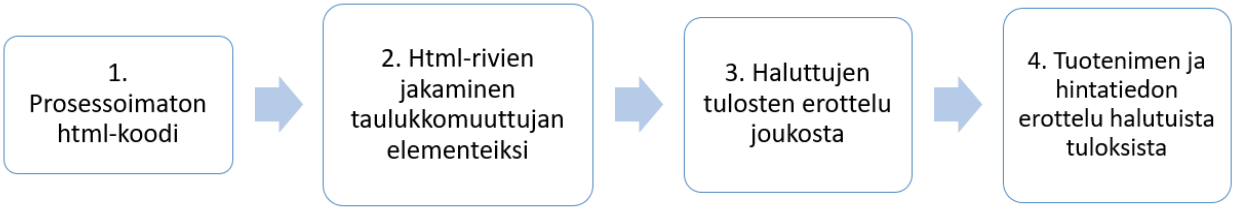
KUVIO 2. HTTP-pyynnön lähettäminen prosessina

Kuviosta 2 käy ilmi HTTP-pyynnön lähettävän funktion toiminta. HTTP/POST-pyyntö muodostettiin PHP/CURL-kirjaston avulla. Pyynnön tiedot annettiin *curl_setopt()*-funktion avulla seuraavasti:

```
curl_setopt($ch, CURLOPT_URL, $url); /* kohde-URL */
curl_setopt($ch, CURLOPT_USERAGENT, 'Testbot'); /* User Agentin määrit-
tely */
curl_setopt($ch, CURLOPT_POST, 1); /* kyseessä on POST-tyypin pyyntö */
curl_setopt($ch, CURLOPT_POSTFIELDS, $data); /* POST-pyynnön sisältö */
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1); /* Tallennetaan vastaus-
viesti muuttujaan */
```

CURLOPT_RETURNTRANSFER-valinnan ansiosta palvelimelta tullut vastaus saatiin tallennettua muuttujaan myöhempää prosessointia varten.

7.3 Halutun informaation poimiminen taulukosta



KUVIO 3. Tiedonerotteluprosessi

Palvelimelta saatu vastaus oli HTML-muotoinen taulukko. HTML-taulukko ei sellaisenaan kelvannut tietokantaan tallennettavaksi. Sisältö täytyi prosessoida oikeaan muotoon. Tiedonerotteluprosessi on kuvattu kuviossa 3. Yritystä kiinnostivat kaikki kulloinkin saatavilla olevat kuukausittaiset, kvartaali- sekä vuosittaiset ENO- ja SYHEL-alkuiset tuotteet. Tuotenimet noudattivat kaavaa:

1) ENOMJAN-17, ENOMFEB-17, ENOMMAR-17, jne.
2) ENOMQ1-17, ENOMQ2-17, ENOMQ3-17, ENOMQ4-17, ENOMQ1-18, jne.
3) ENOMYR-17, ENOMYR-18, jne.
4) SYHELJAN-17, SYHELFEB-17, SYHELMAR-17, jne.
5) SYHELQ1-17, SYHELQ2-17, SYHELQ3-17, SYHELQ4-17, SYHELQ1-18, jne.
6) SYHELYR-17, SYHELYR-18, jne.

Taulukosta oli eroteltava näiden tuotteiden tuotenimet sekä niiden Daily Fix -hinnat tallennusta varten. Erottelua varten oli muodostettava säännölliset lausekkeet, joiden avulla voitiin etsiä haluttuja tuotenimiä. Tuotenimiä ja kohdetekstiä tutkimalla ja analysoimalla voitiin havaita tiettyjä yhtäläisyyksiä nimien välille.

Yksi vaihtoehto olisi luoda kaikki halutut tuotenimikombinaatiot. Tällöin kuukausituotteita varten olisi tarvinnut *date()*-funktion ja *mktime()*-funktioiden avulla määrittää kuluva kuukausi ja seuraavat kuusi kuukautta sekä oikea vuosiluku. Vuosituotteita varten olisi pitänyt määrittää kuluva vuosi sekä neljä tulevaa vuotta. Kvartaalituotteiden nimien määrittäminen olisi ollut kaikista monimutkaisinta. Oikeaa kvartaalia ei saa suoraan mistään funktiosta. Kvartaalinumerointi olisi voitu hoitaa vaikkapa if-lauseilla kuukauden avulla tarkastaen. Mikäli kuluva kuukausi olisi tammi-, helmi- tai maaliskuu, olisi kvartaalin numero 1, jos huhti-, touko- tai kesäkuu, olisi kvartaalin numero 2, ja niin edelleen. Mikäli viimeisimmän muodostetun tuotenimen kvartaalin numero olisi 4, tulisi seuraavan kvartaalin numero olla 1, ja vuosilukua tulisi myös kasvattaa yhdellä. Tällä tavoin muodostettujen tuotenimien säännöllisten lausekkeiden määritelmät rakennettaisiin PHP-koodissa seuraavalla tavalla:

1) `'/ENOM' . $month . '-' . $year . '/i'`, missä `$month` ja `$year` määräytyvät for-silmukalla alkaen kuluvaa seuraavasta kuukaudesta.
2) `'/ENOQ' . $quarter . '-' . $year . '/'`, missä `$quarter` ja `$year` määräytyvät if-lausekkeen avulla kuluvasta kuukaudesta riippuen alkaen seuraavasta alkavasta kvartaalista.
3) `'/ENOYR-' . $year . '/'`, missä `$year` määräytyy for-silmukalla alkaen kuluvaa seuraavasta vuodesta.

4) `'/SYHEL' . $month . '-' . $year . '/i'`, missä `$month` ja `$year` määräytyvät for-silmukalla alkaen kuluvaa seuraavasta kuukaudesta.
5) `'/SYHELQ'. $quarter . '-' . $year . '/'`, missä `$quarter` määräytyy if-lausekkeen avulla kuluvasta kuukaudesta riippuen ja `$year` määräytyy for-silmukalla alkaen kuluvasta vuodesta.
6) `'/SYHELYR-' . $year . '/'`, missä `$year` määräytyy for-silmukalla alkaen kuluvaa seuraavasta vuodesta.

   joissa

   - `$month` muodossa kuukauden englannin kielisen nimen kolme ensimmäistä kirjainta
   - `$year` muodossa vuosiluvun kaksi viimeistä numeroa
   - `$quarter` 1, 2, 3 tai 4
   - `/i` regex-lausekkeen lopussa kertoo, ettei kirjaimien koolla ole merkitystä. Tämä on tarpeellista siksi, että kuukauden nimi tulee *date()*- ja *mktime()*-funktioilta pienillä kirjaimilla kirjoitettuna, ja kohdetekstissä se on kirjoitettu isoilla kirjaimilla.

Toinen vaihtoehto oli määrittää tuotteiden nimet vain niin tarkasti, kuin tarpeen ilman, että vääriä tuotteita tulee valituksi. Parhaassa tapauksessa pystyttäisiin välttämään kuukausien ja vuosilukujen määrittäminen sekä kvartaalinumerointi. Syötettä tutkimalla ja koodia testaamalla kävi ilmi, että seuraavat määritelmät antoivat tismalleen samat hakutulokset, kuin kokonaisten tuotenimien käyttäminen:

1) `'/ENOM/'`
2) `'/ENOQ/'`
3) `'/ENOYR/'`
4) `'/SYHEL' . $month . '/i'`, jossa `$month` määräytyy for-silmukan avulla, alkaen seuraavasta kuukaudesta
5) `'/SYHELQ/'`
6) `'/SYHEYR/'`

Tällä tavoin tarvitsi käyttää selkeästi vähemmän funktioita ja silmukoita sekä säännöllisiä lausekkeita. Koodista voitiin jättää pois vuoden määritys päivämääräfunktioiden avulla, kvartaaleiden määritys if-lausekkeiden avulla sekä eri kuukausi-, vuosi- ja kvartaaliehtojen muodostaminen for-silmukoilla. Säännöllisten lausekkeiden määrä laski 39:stä yhdeksään. Tämä vähensi koodin monimutkaisuutta ja helpotti

luettavuutta. Lisäksi säännölliset lausekkeet olivat edelleen tarkasti rajattuja eivätkä sisältäneet ylimääräistä prosessointia. Ylimääräistä prosessointia olisi ollut tuotekoodien loppuosien tarkastaminen. *Prce_grep()* ei palauta lauseketta vastaavaa tekstijonoa, vaan koko sen taulukkoelementin, josta osuma löytyi. Osuman loppuosalla ei ole merkitystä, sillä haluttiin löytää kaikki yhdistelmät, joilla oli jokin näistä alkuosista. Loppuosien tarkastaminen olisi lisännyt säännöllisen lausekkeen tarkastamiseen kuluvaa aikaa, mutta ei olisi juuri parantanut tarkistustarkkuutta. Kaikki säännölliset lausekkeet tallennettiin taulukko-muuttujaan myöhempää käyttöä varten.

Alla oleva esimerkissä näkyy HTML-taulukko, joka oli alun perin tallennettuna string-muuttujaan tekstijonona.

```
<tr id="derivatesNordicTable-NOPENOYR-20" title="ENOYR-20 - Electricity
Nordic DSFuture">
  <td name="nm" class="text" style="text-align:left;">ENOYR-20</td>
  <td name="bp" class="dec">20.95</td>
                ...
</tr>
<tr id="derivatesNordicTable-NOPENOYR-21" title="ENOYR-21 - Electricity
Nordic DSFuture">
  <td name="nm" class="text" style="text-align:left;">ENOYR-21</td>
  <td name="bp" class="dec">22.49</td>
                ...
</tr>
```

Ennen tietojen rajaamista haluttujen tuotenimien perusteella täytyi html-taulukko muuntaa sellaiseen muotoon, että koodissa olisi helppo tunnistaa tuotenimi ja juuri sille kuuluva hintatieto. Tätä varten tarvitsi jakaa html-taulukko osiin rivejä merkitsevien <tr>-tunnisteiden perusteella. Tehtävä voitiin suorittaa hyvin yksinkertaisesti *preg_split()*-funktiota käyttämällä. *Preg_split()*-funktiolla annettiin parametreiksi ’/</tr>/’ sekä string-muuttuja, joka sisälsi halutun html-taulukon tekstimuodossa. Näin funktio palautti taulukon, jossa jokainen elementti sisälsi yhden html-taulukkorivin.

```
Array
(
    [0] =>
   <tr  id="derivatesNordicTable-NOPENOYR-20"  title="ENOYR-20  -
Electricity Nordic DSFuture">
      <td  name="nm"  class="text"  style="text-align:left;">ENOYR-
20</td>
      <td name="bp" class="dec">20.95</td>
                    ...
    [1] =>
    <tr  id="derivatesNordicTable-NOPENOYR-21"  title="ENOYR-21  -
Electricity Nordic DSFuture">
      <td  name="nm"  class="text"  style="text-align:left;">ENOYR-
21</td>
      <td name="bp" class="dec">22.49</td>
                    ...
)
```

Tämän jälkeen tulosjoukko voitiin rajata vain niihin tuloksiin, jotka sisältävät halutun tuotenimen, ja sen jälkeen halutuista tuloksista voitiin poimia sekä tuotenimi, että sille kuuluva hinta. Tulosjoukon karsinnassa käytettiin *preg_grep()*-funktiota for-silmukan sisälle sijoitettuna. *Preg_grep()* hyväksyy parametreiksi string-muotoisen säännöllisen lausekkeen ja array-muotoisen syötteen. Funktiolle voitiin antaa *preg_split()*-funktiolla array-muotoon jaettu syöte, sekä for-silmukan avulla tuotenimien säännölliset lausekkeet yksi kerrallaan.

Osumat tallennettiin taulukkomuuttujaan *array_merge()*-funktion avulla. Näin voitiin varmistaa, että osumat sisältävä taulukko oli yksiulotteinen. Mikäli osumat olisi tallennettu taulukkomuuttujaan ilman *array_merge()*-funktiota, olisi taulukosta tullut moniulotteinen. Kuvasta 7 voidaan nähdä, että jokainen iteraatio, eli jokaisella eri säännöllisellä lausekkeella saadut osumat, olisi tallennettu uuteen taulukkoelementtiin. Mikäli samalla säännöllisellä lausekkeella saatiin useita tuloksia, tallennettiin ne taulukkomuodossa osumat sisältävään taulukkoelementtiin. Tämä olisi tuottanut ongelmia seuraavassa vaiheessa, kun osumista lähdettiin erottelemaan tuotekoodeja ja hintatietoja tietokantaan syöttämistä varten.

pcre_grep() iteraatiot	Ilman array_merge()-funktiota	Array_merge()-funktiolla
1. iteraatio	Array(key => value,	Array(key => value,
2. iteraatio	key1 => array(key1 => value1, key2 => value2, key3 => value3)	key1 => value1, key2 => value2, key3 => value3,
3. iteraatio	key2 => value4,	key4 => value4,
4. iteraatio	key3 => array(key5 => value5, key6 => value6));	key5 => value5, key6 => value6);

KUVA 7. *Pcre_grep()*-funktion tulosten tallentaminen taulukkoon *array_merge()*-funktiolla ja ilman

Tulosten rajaamisen jälkeen kaikista osumista, eli html-taulukkoriveistä, jotka sisälsivät haluttujen tuotteiden tiedot, eroteltiin tuotteiden nimet ja Daily Fix-hinnat tietokantaan syöttämistä varten. HTML-taulukkoa tarkastelemalla voitiin huomata, että tuotenimi oli halutussa muodossa aina ”derivatesNordicTable-NOP”-tekstinpätkän jälkeen, ja päättyi ”-merkkiin. Nämä voidaan ilmaista säännöllisessä lausekkeessa alilausekkeena. Alilausekkeet otetaan huomioon lauseketta vastaavia tuloksia etsittäessä, mutta sitä ei sisällytetä tulosta tulostettaessa. Ennen varsinaista säännöllistä lauseketta esiintyvä, tuloksesta pois jätettävä alilauseke merkitään `(?<=)` -rakenteen sisälle. Ennen haluttua tuotekoodia tulevaa ”derivatesNordicTable-NOP”-tekstiä merkittäisiin lausekkeella `(?=<derivatesNordicTable-NOP)`. Varsinaisen säännöllisen lausekkeen jälkeen tuleva, tuloksesta pois jätettävä alilauseke merkitään `(?=)` -rakenteen sisään. Tuotekoodin jälkeen esiintyvää ”-merkkiä merkittäisiin lausekkeella `(?=”)`.

Hintatietojen erottelu onnistuu samalla tavalla. Taulukon otsikkoriviä kuvaavasta osasta voitiin nähdä, että Daily Fix-hinnan data-atribuutti taulukossa oli ”stlpr”. Taulukkoa tutkimalla voitiin havaita, että oikea hintatieto pystyttiin kuvaamaan säännöllisellä lausekkeella `/stlpr" class="dec">[0-9]{1,2}\.[0-9]{1,2}</`. Tällä lausekkeella hakutulos olisi ollut muodossa `’stlpr" class="dec">0.00<’`, jossa nollien paikalla voisi olla mikä tahansa desimaaliluku, jossa on enintään kaksi lukua ennen ja jälkeen desimaalipisteen. Koska haluttiin tulostaa vain hintaa kuvaava lukuarvo, piti lukuarvoa ennen ja sen jälkeen tulevat osat ilmaista alilausekkeilla. Lauseke `/(?<=stlpr"`

`class="dec">)[0-9]{1,2}\.[0-9]{1,2}(?=<)/` täsmää saman merkkijonon, mutta tulostaa vain hintatiedon.

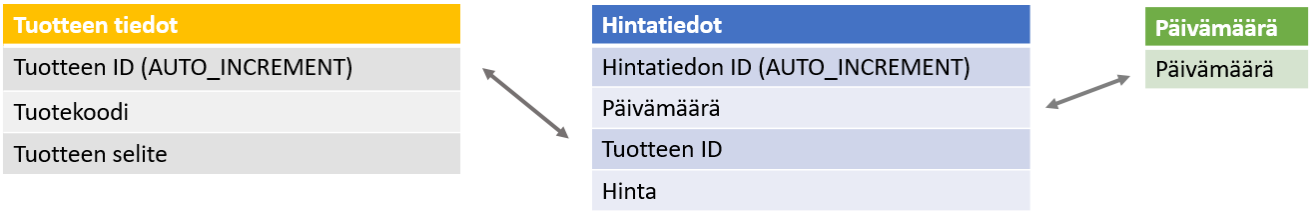
Lopuksi saadut nimi- ja hintatiedot tallennettiin taulukkomuuttujaan avain-arvo -pareina.

```
Array
(
    [0] => Array
        (
            [ENOMAPR-17] => 26.10
        )

    [1] => Array
        (
            [ENOMMAY-17] => 22.60
        )

    [2] => Array
        (
            [ENOMJUN-17] => 22.10
        )
…
)
```

Päivämäärä haettiin tulosjoukon ensimmäisestä solusta, joka sisälsi taulukon otsikkotietoja. Päivämäärä esiintyy siellä aina samassa kohtaa ja samassa muodossa. Niinpä sille voitiin helposti muodostaa säännöllinen lauseke. Kuten tuotekoodien ja hintatietojenkin tapauksessa, säännöllisen lauseen määrittelyssä käytettiin hyväksi alilausekkeita.

7.4 Tietojen syöttäminen tietokantaan

Tietojen syöttäminen tietokantaan oli aloitettava tietokannan tarkastelulla. Oikeanlaisten SQL-kyselyiden muodostamisen kannalta oli tärkeä tietää, mihin tauluun ja minkä nimiseen sarakkeeseen mikäkin tieto oli syötettävä ja missä järjestyksessä. Kyseessä olevan Daily Fix -hintatiedon osalta tietokantaan halutaan kerätä mahdollisimman laajasti tietoa. Edellisten päivien tietojen säilyttäminen on tärkeää hintatietojen käyttäytymisen analysoinnin kannalta, joten aikaisemmin tallennettujen tietojen poistamiselle ei ollut skriptissä tarvetta.



KUVA 8. Tietokannan taulujen rakenne

Tiedot syötetään kolmeen eri tauluun. Kuvassa 8 näiden taulujen nimet on kirjoitettu valkoisella, ja mustalla kirjoitetut rivit kuvaavat taulukon kenttiä. Harmaat nuolet osoittavat yhteyksiä taulujen välillä. Ensimmäiseksi täytyy tallentaa päivämäärä-tauluun päivämäärä sekä tuotteen tiedot -tauluun tuotteen ID ja selite, sillä ilman niitä tietojen lisääminen hintatietoja varastoivaan tauluun ei onnistu. Tuotteiden tietoja sisältävään tauluun lisätään vain sellaiset tuotteet, joita siellä ei vielä ole. Hintatiedot-taulun INSERT-kyselyä varten tarvitsee tietää lisättävän tuotteen ID. Jotta voitaisiin saada kulloinkin käsiteltävän tuotekoodin ID tietoon, on lähetettävä tuotetietoja sisältävään tauluun SELECT-kysely. SELECT-kyselyllä haetaan sen tuotteen ID:tä, jonka tuotekoodi vastaa käsiteltävää tuotekoodia. Mikäli ID löytyy, se tallennetaan muuttujaan. Jos ID:tä ei löydy, lähetetään tuotekoodin tiedot tietokantaan, ja haetaan sen jälkeen juuri lisätyn tietueen ID *mysqli_insert_id()*-funktion avulla.

Tarvittavat SQL-kyselyt olivat:

- Päivämäärän lisääminen: `INSERT INTO taulu1 (arvo) VALUES ('$arvo')`
- Tuotteen ID:n hakeminen: `SELECT arvo1 FROM taulu2 WHERE arvo2 = '$arvo2'`
- Tuotekoodin tietojen lisääminen: `INSERT INTO taulu2 (arvo1, arvo2) VALUES ('$arvo1', '$arvo2')`
- Hintatiedon lisääminen: `INSERT INTO taulu3 (arvo1, arvo2, arvo3) VALUES ('$arvo1', '$arvo2, '$arvo3')`

Tuotekoodin lisäksi tarvitaan vielä tuotteen selite, ennen kuin tuotekoodin tietoja voidaan lisätä tietokantaan. Tuotteen selite määritellään jokaiselle tuotteelle erikseen. Tämä prosessi suoritetaan vain silloin, kun uusia tuotteita lisätään. Suurimmalla osalla ajokerroista uusia tuotteita ei täydy lisätä, joten useimmiten tätä prosessia ei tarvitse suorittaa ollenkaan. Määrittelyprosessi on toteutettu if- sekä switch-lauseilla ja hyödyntää *preg_match()*-funktiota. Tuotteiden selitteet muodostuvat seuraavalla tavalla:

1. Mikäli kyseessä on SYHEL-tuote, lisätään selitteen eteen ”Aluehinta”
2. Mikäli tuotekoodi sisältää kirjainyhdistelmän ”YR”, on kyseessä vuosituote

3. Mikäli tuotekoodi sisältää kirjaimen ”Q”, on kyseessä kvartaalituote.
    a. Tällöin selitteeseen lisätään ”Vuosituote”
    b. Etsitään Q-kirjaimen jälkeen esiintyvä numero ja lisätään sen perusteella tuoteselitteen jatkoksi ”/ $numero. kvartaali”, missä $numero korvataan kvartaalin numerolla
4. Mikäli tuotekoodi ei sisällä ”YR”- eikä ”Q”- kirjaimia, voidaan olettaa kyseessä olevan kuukausituote
    a. Tällöin selitteeseen lisätään ”Kuukausituote”
    b. Etsitään ennen tuotekoodissa ennen väliviivaa esiintyvät kolme kirjainta ja lisätään selitteen jatkoksi oikea suomenkielinen kuukaudennimi. Oikea kuukaudennimi määritellään switch-rakenteella
5. Lopuksi etsitään tuotekoodissa väliviivan jälkeen olevat kaksi numeroa ja lisätään tuoteselitteen loppuun ”20$vuosi”, missä $vuosi on väliviivan jälkeen esiintyvät kaksi numeroa.

Esimerkiksi:

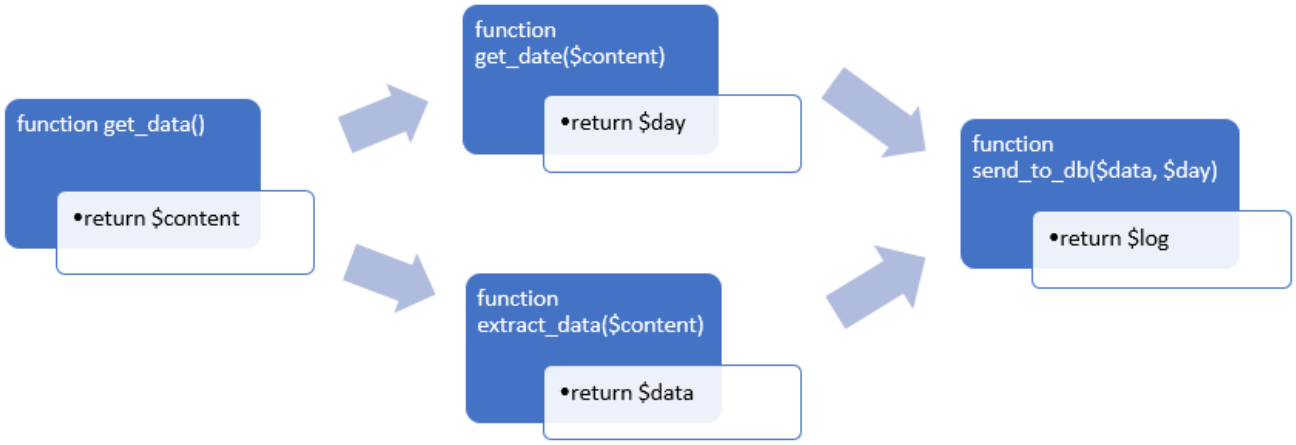
SYHELJAN-18	= Aluehinta Kuukausituote Tammikuu 2018
ENOQ1-22	= Vuosituote / 1. kvartaali 2022
SYHELYR-20	= Aluehinta Vuosituote 2020

Tuotetietojen lähettäminen sujuu edellä selostetun prosessin mukaisesti jokaiselle tuotekoodille, joten prosessi on järkevä toteuttaa silmukkana. Koska tuotekoodit ja hinnat on tallennettu taulukkoon, voidaan käyttää foreach-silmukkaa, joka käy kaikki taulukon rivit yksitellen läpi. Päivämäärä lisätään päivämäärä-tauluun vain kerran ja se on sama kaikille tuotteille, joten se voidaan tehdä ennen tuotetietoja lähettävää foreach-silmukkaa.

Jokaisen suoritetun tietokantakyselyn yhteydessä lisätään lokiviestiin rivi, josta käy ilmi, millainen SQL-kysely on lähetetty. Lisäksi lisätään tieto siitä, oliko käskyn ajo onnistunut, vai päättyikö se vikailmoitukseen. Lokin avulla käyttäjä voi jälkikäteen tarkistaa, miten ajo sujui. Tämä on kätevää esimerkiksi silloin, kun ohjelma ajetaan ajastuspalvelun avulla, eikä ihminen ole suoritusajankohtana valvomassa ohjelman toimintaa. Viesti voitaisiin myös lähettää käyttäjälle vaikkapa sähköpostina, jolloin käyttäjän ei tarvitsisi erikseen kirjautua palvelimelle tarkastamaan lokitiedostoa. Esimerkki lokiviestin tulostuksesta nähtävissä alla.

```
vvvv-kk-pp saved successfully
INSERT INTO taulu1 (arvo1, arvo2, arvo3) VALUES ('arvo1', 'arvo2',
'arvo3') saved successfully
INSERT INTO taulu2 (arvo1, arvo2) VALUES ('arvo1', 'arvo2') saved
successfully
INSERT INTO taulu1 (arvo1, arvo2, arvo3) VALUES ('arvo1', 'arvo2',
'arvo3') saved successfully
INSERT INTO taulu2 (arvo1, arvo2) VALUES ('arvo1', 'arvo2') saved
successfully
```

7.5 Ohjelman rakenne



KUVIO 4. Ohjelman rakenne

Kuvio 4 esittää skriptin rakennetta. Kuten kaaviosta nähdään, edellä selostetut työvaiheet vastaavat pääosin skriptissä omaa funktiotaan. Halutun informaation poimiminen taulukosta on kuitenkin jaettu kahteen osaan, *extract_data()* ja *get_date()*. Funktiot kutsutaan vuorotellen kaaviota 4 mukailevassa järjestyksessä.

Skriptin ensimmäinen funktio hakee kohdesivulta html-taulukon. Tälle funktiolle ei syötetä mitään parametreja. Suoritettaessa se palauttaa kohdesivulta haetun html-taulukon PHP:n taulukkoelementteihin jaettuna. Toinen funktio huolehtii datan prosessoimisesta ja suodattamisesta. Tälle funktiolle annetaan parametrina ensimmäisen funktion tulos. Toinen funktio palauttaa taulukon, johon on tallennettu tuotteiden tuotekoodit ja hinnat. Kolmas funktio ottaa myös parametrikseen ensimmäisen funktion tuloksen. Kolmannen funktion tehtävä on etsiä ja palauttaa tekstistä löytyvä päivämäärä. Neljäs funktio huolehtii

tietojen syöttämisestä tietokantaan. Se ottaa parametreikseen sekä toisen että kolmannen funktion tulokset, eli päivämäärän sekä taulukon, jossa on tuotteiden hintatiedot. Neljäs funktio palauttaa lokiviestin, josta käy ilmi tietokantakyselyiden onnistuminen tai epäonnistuminen.

7.6 Tehtävän ajastus

Tehtävä halutaan suoritettavan joka päivä illalla kello yhdentoista jälkeen. Tällöin tehtävän aikamääritelmä voisi olla esimerkiksi ”05 23 * * *”, eli viisi minuuttia yli iltayhdentoista jokaisen kuukauden jokaisena päivänä. Käsky, jolla PHP-ohjelma suoritetaan, riippuu PHP-ohjelman ja käytettävän Cronin sijainnista. Mikäli Cron ja PHP-ohjelman sijaitsevat samalla palvelimella, voidaan suorituskäskyksi antaa esimerkiksi `php -f /polku/oikeaan/kansioon/php_skripti.php`. Jos taas PHP-skripti sijaitsisi web-palvelimella ja Cron erillisellä ajastuspalvelimella, voitaisiin tehtävän ajo laukaista vierailemalla skriptin verkko-osoitteessa.

Testasin PHP-ohjelman Cron-ajoa Ubuntu-käyttöjärjestelmällä. Määrittelin tehtävän ajastuksen crontabiin seuraavasti:

```
05 23 * * * php -f /polku/oikeaan/kansioon/php_skripti.php > output.txt
```

Tämä käsky suoritti PHP-ohjelman kello 23:05 ja ohjasi ohjelman tulostaman lokiviestin tiedostoon nimeltä output.txt.

8 POHDINTA

Opinnäytetyön tehtävänä oli automatisoida verkkosivun tiedonharavointiprosessi ja testata sen ajastamismahdollisuuksia. Tuloksena oli PHP-ohjelma, joka kykenee itsenäisesti hakemaan, erottelemaan ja tallentamaan tietokantaan halutut tiedot. Ajastamismahdollisuuksia esiteltiin teoriaosassa, ja Cronin käyttöä testattiin Ubuntu-käyttöjärjestelmällä käytännön osuudessa.

Teoriaosuudessa on esitelty käytetyt työkalut sekä hyödylliset käsitteet pääpiirteittäin. Käytännön osuus on dokumentoitu yksityiskohtaisesti kirjalliseen tuotokseen. Jokainen vaihe on esitelty omassa alakappaleessaan. Käytännön osuuden prosesseja ja valintoja on pyritty esittämään sekä kirjallisesti että esimerkein ja prosessikaavioilla tukien.

Työ oli kaiken kaikkiaan melko onnistunut. Haluttu prosessi saatiin automatisoitua ja työn tuloksena syntynyt ohjelman on käyttökelpoinen ja edelleen kehitettävissä. Pääsin työtä tehdessäni käyttämään sekä työkaluja, jotka olivat minulle ennalta tuttuja, että syventymään kokonaan uusien työkalujen käyttöön.

Kokoamaani PHP-ohjelmaa voitaisiin kehittää tarpeen mukaan esimerkiksi rakentamalla sille konfigurointisivu, lisäämällä ajastuksen yhteyteen käyttäjän tunnistus, lisäämällä lokiviestiin käyttäjän identiteetti sekä suoritusajankohta ja lokiviesti voitaisiin ohjata käyttäjän sähköpostiin. Nämä toimenpiteet lisäisivät ohjelman helppokäyttöisyyttä sekä luotettavuutta. Konfigurointisivu auttaisi ei-koodaustaitoista käyttäjää vaihtamaan vaivattomasti asetuksia, kuten HTTP-pyynnön asetuksia tai tietokantaan liittyviä asetuksia. Käyttäjän tunnistaminen auttaisi ehkäisemään epätoivottuja tehtävän suorittamisia, ja tunnistustiedon ja tehtävän suorittamisen ajankohdan ilmoittaminen lokiviestissä auttaisi tunnistamaan, onko suoritus ollut tarkoituksenmukainen vai ei. Lokiviestin ohjaaminen sähköpostiin helpottaisi käyttäjän arkea siten, että ajojen onnistumista olisi helppo seurata. Käyttäjän ei tarvitsisi erikseen kirjautua palvelimelle tarkastamaan ajon onnistumista, vaan hän saisi siitä tiedon välittömästi ja vaivattomasti ajon jälkeen vaikkapa puhelimeensa.

LÄHTEET

Cronless.com. Pricing & Sing up. Internet-sivu. Saatavissa: http://cronless.com/pricing.php. Viitattu 6.4.2017.

Easycron.com. Sign Up. Internet-sivu. Saatavissa: https://www.easycron.com/user/register/. Viitattu 6.4.2017.

EnergiaGuru a. EnergiaGuru – koko sähköalaa mullistava palvelu. Internet-sivu. Saatavissa: https://energiaguru.fi/. Viitattu 17.5.2017.

EnergiaGuru b. Paljon tyytyväisiä asiakkaita. Internet-sivu. Saatavissa: https://energiaguru.fi/referenssit/. Viitattu 17.5.2017.

Glez-Peña, D., Lourenco, A., Lopez-Fernandez H., Reboiro-Jato, M. & Fdez-Riverola, F. 2013. Web scraping technologies in an API world. Artikkeli.

Linux.fi. 2016. Komentojen ajastaminen. Wiki-artikkeli. Saatavissa: https://www.linux.fi/wiki/Komentojen_ajastaminen. Viitattu 6.4.2017

MySQL 5.7 Reference Manual. 2017. Online-ohjekirja. Saatavissa: https://dev.mysql.com/doc/refman/5.7/en/. Viitattu 7.4.2017.

PHP Manual. 2017a. Client URL Library. Online-ohjekirja. Saatavissa: http://php.net/manual/en/book.curl.php. Viitattu 7.4.2017.

PHP Manual. 2017b. MySQL Improved Extension. Online-ohjekirja. Saatavissa: http://php.net/manual/en/book.mysqli.php. Viitattu 7.4.2017.

PHP Manual. 2017c. Regular Expressions (Perl-Compatible). Online-ohjekirja. Saatavissa: http://php.net/manual/en/book.pcre.php. Viitattu 7.4.2017.

RFC 7231. 2014. Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. Internet-julkaisu. Saatavissa: https://tools.ietf.org/html/rfc7231. Viitattu 28.3.2017.

RFC 7230. 2014. Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. Internet-julkaisu. Saatavissa: https://tools.ietf.org/html/rfc7230. Viitattu 28.3.2017.

Schrenk, Michael. 2012. Webbots, Spiders, and Screen Scrapers, 2nd Edition. San Francisco: No Starch Press, Inc.

Ubuntu Documentation, Community Help Wiki. 2016. CronHowto. Wiki-artikkeli. Saatavissa: https://help.ubuntu.com/community/CronHowto. Viitattu 6.4.2017.

Wireshark User's Guide. 2014. Online-ohjekirja. Saatavissa: https://www.wireshark.org/docs/wsug_html_chunked/index.html. Viitattu 6.4.2017.

https://courses.cs.washington.edu/courses/cse190m/12sp/cheat-sheets/php-regex-cheat-sheet.pdf

PHP PCRE Cheat Sheet

Functions
preg_match(pattern, subject[, submatches])
preg_match_all(pattern, subject[, submatches])
preg_replace(pattern, replacement, subject)
preg_replace_callback(pattern, callback, subject)
preg_grep(pattern, array)
preg_split(pattern, subject)

Base Character Classes	
\w	Any "word" character (a-z 0-9 _)
\W	Any non "word" character
\s	Whitespace (space, tab CRLF)
\S	Any non whitepsace character
\d	Digits (0-9)
\D	Any non digit character
.	(Period) – Any character except newline

Meta Characters	
^	Start of subject (or line in multiline mode)
$	End of subject (or line in multiline mode)
[	Start character class definition
]	End character class definition
|	Alternates, eg (a|b) matches a or b
(	Start subpattern
)	End subpattern
\	Escape character

Quantifiers	
n*	Zero or more of n
n+	One or more of n
n?	Zero or one occurrences of n
{n}	n occurrences exactly
{n,}	At least n occurrences
{,m}	At most m occurrences
{n,m}	Between n and m occurrences (inclusive)

Pattern Modifiers	
i	Caseless – ignore case
m	Multiline mode - ^ and $ match start and end of lines
s	Dotall - . class includes newline
x	Extended– comments & whitespace
e	preg_replace only – enables evaluation of replacement as PHP code
S	Extra analysis of pattern
U	Pattern is ungreedy
u	Pattern is treated as UTF-8

Point based assertions	
\b	Word boundary
\B	Not a word boundary
\A	Start of subject
\Z	End of subject or newline at end
\z	End of subject
\G	First matching position in subject

Subpattern Modifiers & Assertions		
(?:)	Non capturing subpattern	((?:foo|fu)bar) matches foobar or fubar without foo or fu appearing as a captured subpattern
(?=)	Positive look ahead assertion	foo(?=bar) matches foo when followed by bar
(?!)	Negative look ahead assertion	foo(?!bar) matches foo when *not* followed by bar
(?<=)	Positive look behind assertion	(?<=foo)bar matches bar when preceded by foo
(?<!)	Negative look behind assertion	(?<!foo)bar matches bar when *not* preceded by foo
(?>)	Once-only subpatterns	(?>\d+)bar Performance enhancing when bar not present
(?(x))	Conditional subpatterns	(?(3)foo|fu)bar Matches foo if 3rd subpattern has matched, fu if not
(?#)	Comment	(?# Pattern does x y or z)